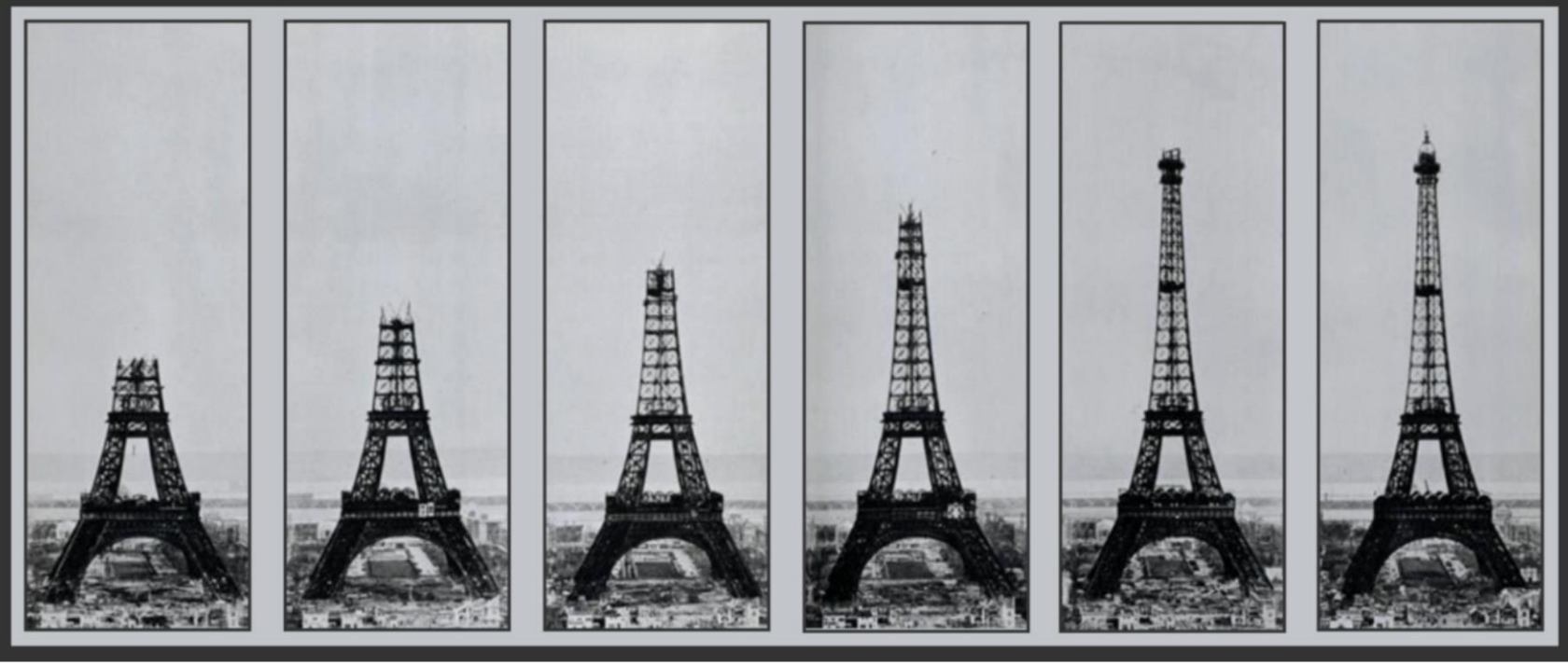
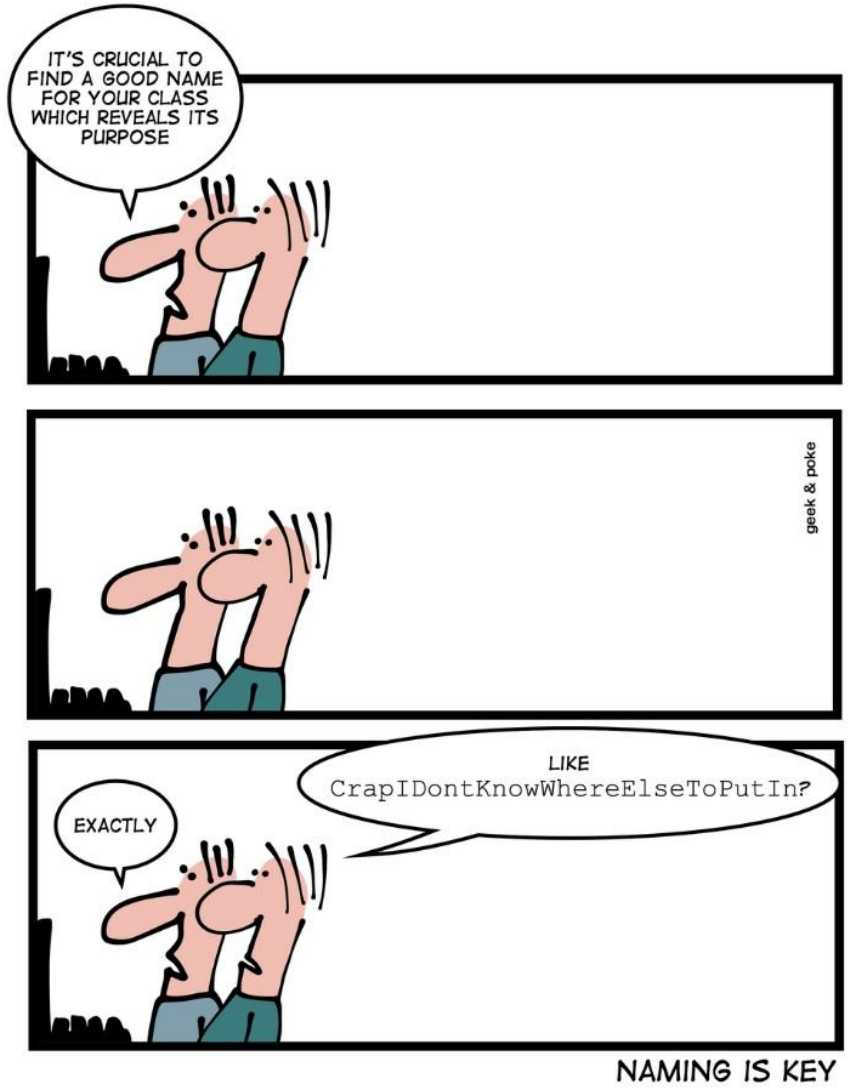
Structure d’un programme



Antoine SCHMID, CFPT-Informatique
Avec la collaboration du CPLN

Table des Matières

- Résumé du chapitre précédent
- Structure d’un programme
- Variables et types



Dans l’épisode précédent…

- **Informatique** = INFormations automaTIQUE.
- Ordinateur, instructions, code source
- Algorithme

- **JavaScript**
- ECMAScript
- Hello, World !

Structure d’un programme

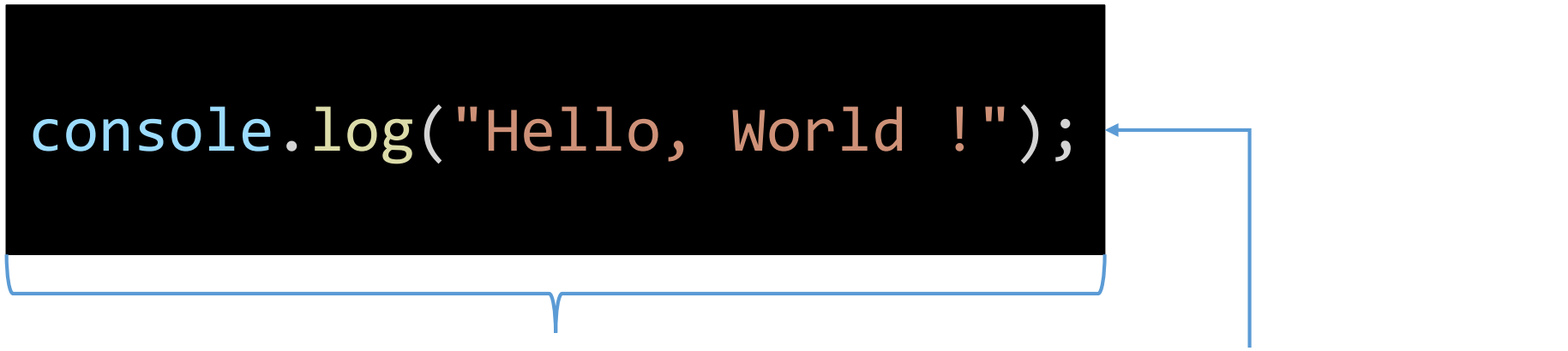
- Nous avons précédemment défini un **programme informatique** comme étant une liste d'ordres indiquant à un ordinateur ce qu'il doit faire.

- Ces ordres sont écrits sous forme de texte dans un ou plusieurs fichiers et forment ce qu'on appelle le code source du programme.

- Les lignes de texte dans un fichier de **code source** s'appellent des **lignes de code**.

Instructions

- Chaque ordre inclus dans un programme est appelée une **instruction**.
- Une instruction est délimitée par un **point virgule**.
- Un programme est constitué d'une suite d'instructions.



Commentaires

- Chaque ligne de texte dans les fichiers source d'un programme est considérée comme une instruction à exécuter.

- Il est possible d'exclure certaines lignes de l'exécution en les préfixant par une double slash **//**.

- On transforme ces lignes en **commentaires**.

- Il existe une autre manière de créer des commentaires en entourant une ou plusieurs lignes par les caractères **/*** et ***/**.

Commentaires

```
// Un commentaire sur une seule ligne
```

```
/* Un commentaire
   sur plusieurs
   lignes */
```

Commentaires

- Les commentaires sont des indications placées au milieu d'un script et servant à **documenter le code**, c'est-à-dire à expliquer ce que fait tel ou tel bout de script et éventuellement comment le manipuler.

- Commenter va servir aux développeurs à se repérer plus facilement dans un script, à le lire et à le comprendre plus vite.

- Cela peut être utile à la fois pour vous même si vous travaillez sur des projets complexes ou pour d'autres développeurs si vous êtes amené à distribuer votre code un jour ou l'autre.

Indentation

- **L’indentation** correspond au fait de **décaler** certaines lignes de code par rapport à d’autres.
- Cela est généralement utilisé pour rendre son code plus lisible et donc plus simple à comprendre.

- La norme en JavaScript est d’effectuer un retrait vers la droite équivalent à 2 espaces à chaque fois qu’on écrit une nouvelle ligne de code à l’intérieur d’une instruction. Nous aurons l’occasion d’illustrer cela plus tard.

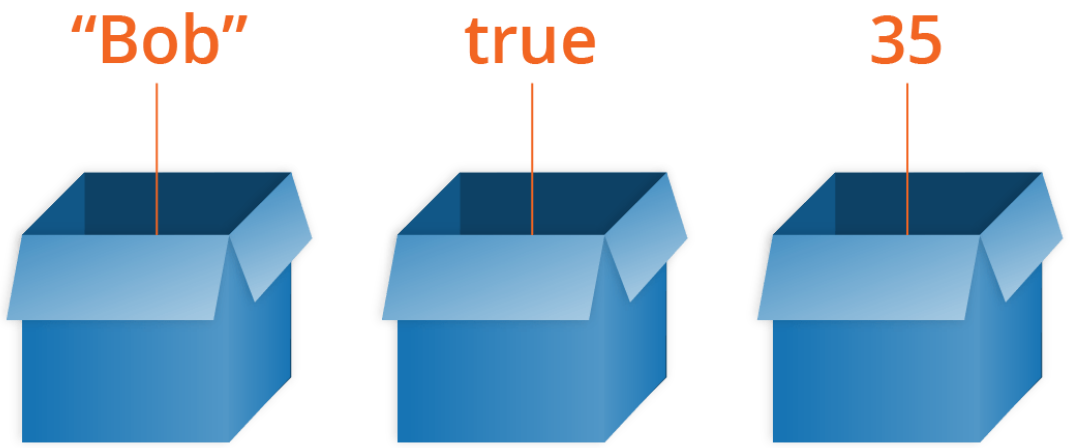
Exercices

- Exercice 2.1 : **Exercices de base**

Variables et types

Rôle des variables

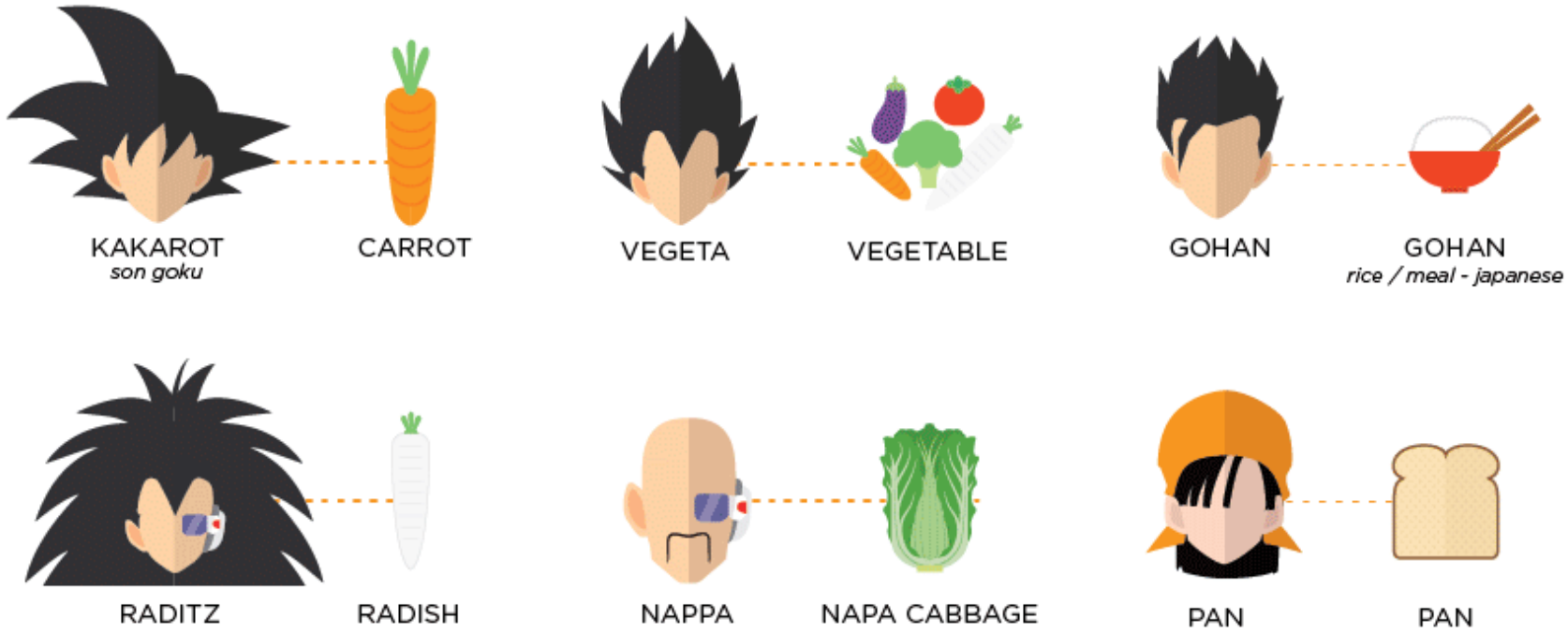
- Un programme informatique mémorise des données en utilisant des variables.
- Une **variable** est une zone de stockage d'information.
- On peut l'imaginer comme une boîte dans laquelle on range des choses.



Propriétés d’une variable

- Une variable possède trois grandes propriétés :
  - Son **type**, qui détermine le rôle et les opérations applicables à cette variable.
  - Son **nom**, qui permet de l'identifier.
  - Sa **valeur**, qui est la donnée actuellement mémorisée dans cette variable.

Noms


KAKAROT
son goku
CARROT
VEGETA
VEGETABLE
GOHAN
GOHAN
rice / meal - japanese
RADITZ
RADISH
NAPPA
NAPA CABBAGE
PAN
PAN

Nom

- Le **nom** (ou le label) d'une variable permet de
  - Retrouver notre variable dans la mémoire
  - Savoir qui on manipule

- En JavaScript, un nom de variable peut contenir
  - des lettres majuscules ou minuscules
  - des chiffres (sauf en première position)
  - et certains caractères comme le dollar ($) ou le tiret bas (*underscore*) (_)

Nom – normes

- En JavaScript, la norme pour une variable est d'utiliser la notation **lowerCamelCase**
  - Les mots sont écrits en minuscules
  - Séparés par des majuscules
  - Ex: `munitionsPistolet, ageCapitaine, …`

- Si la variable est **constante**, on utilisera la notation **CONSTANT_CASE**
  - Les mots sont écrits en majuscules
  - Séparés par des underscores
  - Ex: `VITESSE_VERTICALE, PI, …`

Nom – à ne pas faire…

- Mettre des accents !
  `durée, âgeCapitaine`
- Mettre une **majuscule** au début des variables
  `MyVar, Health`
- Les variables **trop longues**
  `superLongueVariableDeLaMortQuiTue`,
  `super_longue_variable_de_la_mort_qui_tue`.
- Les variables qui sont **incompréhensibles**
  `lVar` (au lieu de **`longueurVar`**)

Nom

- Le nom choisi pour une variable n'a pour la machine aucune importance, et le programme fonctionnera de manière identique.

- Rien n'empêche de nommer toutes ses variables **a**, **b**, **c**..., voire de choisir des noms absurdes comme **steackhache** ou **jesuisuncodeurfou**.

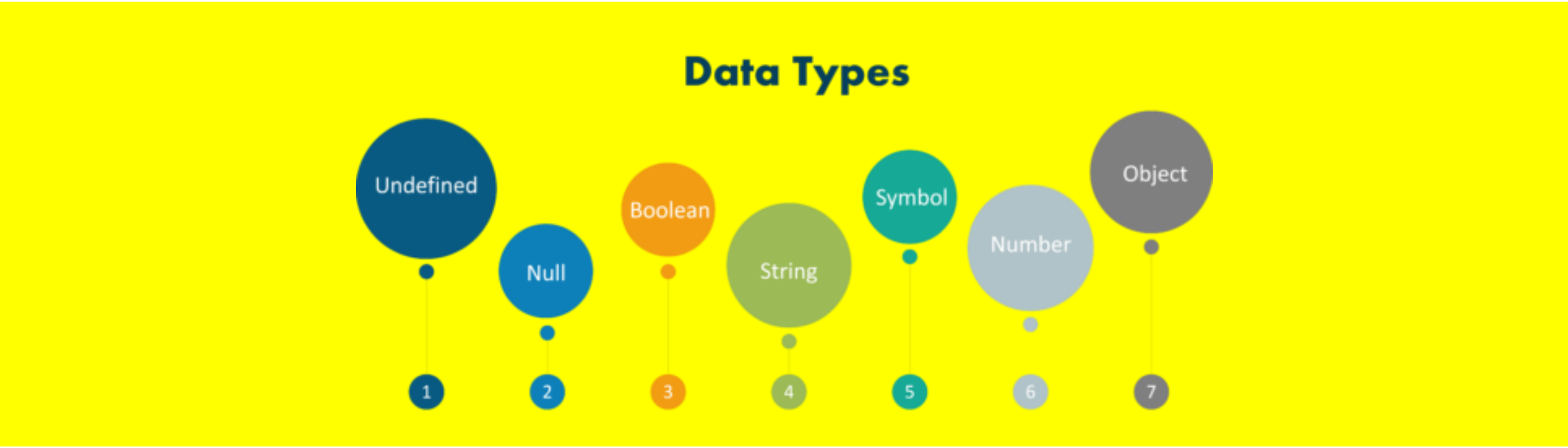
- Mais…

Nom

```
const nb1 = 5.5;
const nb2 = 3.14;
const nb3 = 2 * nb2 * nb1;
console.log(nb3);
```


```
const rayon = 5.5;
const pi = 3.14;
const perimetre = 2 * pi * rayon;
console.log(perimetre);
```

Types


Data Types
Undefined
Null
Boolean
String
Symbol
Number
Object
1
2
3
4
5
6
7

Types

- En JavaScript, contrairement à d'autres langages de programmation, nous n'avons pas besoin de préciser le type de valeur qu'une variable va pouvoir stocker.

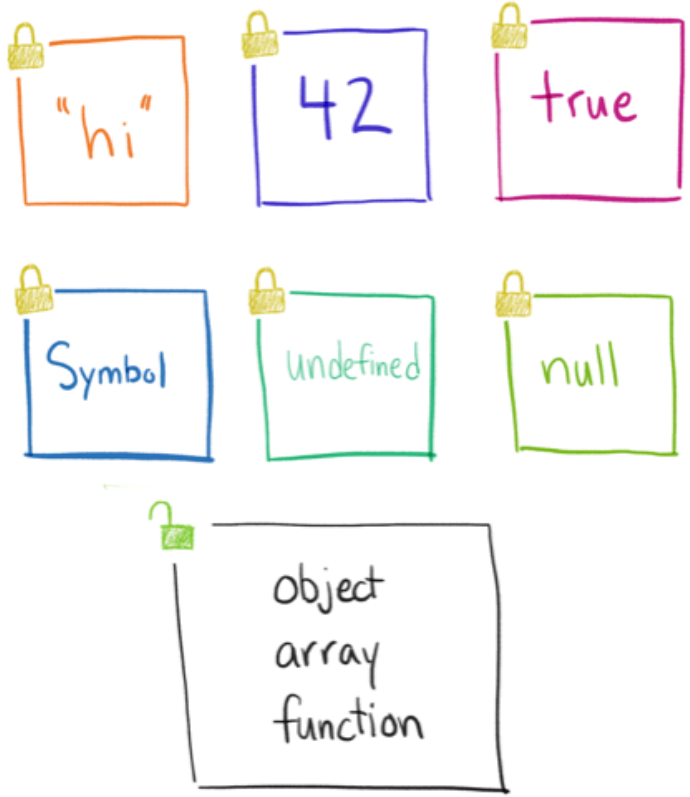
- Le type est automatiquement détecté par l'interpréteur

- En fonction du type identifié, il sera possible d'effectuer certaines opérations (ou non…)

- **Attention**: cela veut dire qu'une variable va pouvoir stocker une valeur textuelle à un moment dans un script puis un nombre à un autre moment.

Types

- En JavaScript, il existe 7 types différents
  - **`String`** ou « chaine de caractères » en français
  - **`Number`** ou « nombre » en français
  - **`Boolean`** ou « booléen » en français
  - **`Null`** ou « nul / vide » en français
  - **`Undefined`** ou « indéfini » en français
  - **`Symbol`** ou « symbole » en français
  - **`Object`** ou « objet » en français



String – chaine de caractères

- Une chaine de caractères et une séquence de caractères => un texte !

- Les valeurs sont délimitées par une paire de guillemets simples ou doubles

```
let maChaine = "Ceci est une chaîne";
let secondExemple = 'Ceci est une chaîne';
let douze = "12";
```

String – chaine de caractères

- Pour inclure des caractères spéciaux, on utilise le caractère \ (*backslash* ou *antislash*)
  - Ex: **\n** permet d'ajouter un retour à la ligne dans une chaine

- On ne peut pas additionner ou soustraire des string comme on peut le faire avec des nombres.
- L'opérateur **+** peut être appliqué à deux string. Son résultat est la jointure de ces deux chaînes, appelée **concaténation**.
  - Ex: `"Bon" + "jour"  →  "Bonjour"`

Number – nombre

- En JavaScript, et contrairement à la majorité des langages, il n'existe qu'un type prédéfini qui va regrouper tous les nombres qu'ils soient positifs, négatifs, entiers ou décimaux (à virgule) et qui est le type **Number**.

```
let douze = 12;
let pi = 3.1415;
let temperature = -16;
```

Pas de guillemets

Le séparateur pour les décimaux est le point .

Number – nombre

- Nous pouvons appliquer à des valeurs de type nombre les mêmes opérations qu'en mathématiques.
- Ces opérations produisent un résultat lui aussi de type nombre

Opérateur	Rôle	Exemple
+	Addition	`3 + 2 = 5`
-	Soustraction	`3 – 2 = 1`
*	Multiplication	`3 * 2 = 6`
/	Division	`3 / 2 = 1.5`
%	Reste (modulo)	`3 % 2 = 1`

Boolean – booléen

- Un booléen, en algèbre, est une valeur binaire (soit 0, soit 1).
- En informatique, le type booléen est un type qui ne contient que deux valeurs : les valeurs `true` (vrai) et `false` (faux)

```
let vrai = true;
let faux = false;

let test = 8 > 4;
```

Null et Undefined

- La valeur `null` correspond à l'absence de valeur connue.
- Pour qu'une variable contienne `null`, il va falloir stocker cette valeur de manière explicite.

```
let resultat = null;
```

- La valeur `undefined` correspond à une variable « non définie », c'est-à-dire une variable à laquelle on n'a pas affecté de valeur

```
let resultat;
```

Object et Symbol

- Les objets sont des structures complexes qui vont pouvoir stocker plusieurs valeurs en même temps
- Nous les étudierons plus tard !

- Les symboles aussi seront vu plus tard

Déclaration des variables

Déclaration des variables

- Avant de pouvoir stocker des informations dans une variable, il faut la créer.
- Cette opération s'appelle la **déclaration** de la variable.

- La déclaration est nécessaire pour
  - Réserver une zone de la mémoire attribuée à cette variable.
  - Lire ou écrire des données dans cette zone mémoire en manipulant la variable.

Déclaration des variables

- Pour déclarer une variable en JavaScript, il faut utiliser le mot clé **`let`**

```
let nom;
let prenom;
let age;
```

- A noter que les anciennes versions de JavaScript utilisaient le mot clé **`var`** qui ne devrait plus être utilisé aujourd'hui !

Déclaration des variables constantes

- Si la valeur initiale d'une variable ne changera jamais au cours de l'exécution du programme, cette variable est ce qu'on appelle une **constante**.
- Il faut la déclarer avec le mot-clé **`const`**
- Cela rend le programme plus facile à comprendre et cela permet de détecter des erreurs

```
const PI = 3.1415;
PI = 5.7;  // ERROR: Assignment to constant variable
```

Affectation des variables

Affectation des variables

- Au cours du déroulement du programme, la valeur stockée dans une variable peut changer.
- Pour donner une nouvelle valeur à une variable,
  on utilise l'opérateur **=**, appelé **opérateur d'affectation**.

```
let nom;
let prenom;

nom = "Schmid";
prenom = "Monsieur";
```

Opérateurs d'affectation

- Il existe d'autre **opérateurs d'affectation** qui permettent de manipuler la valeur d'une variable

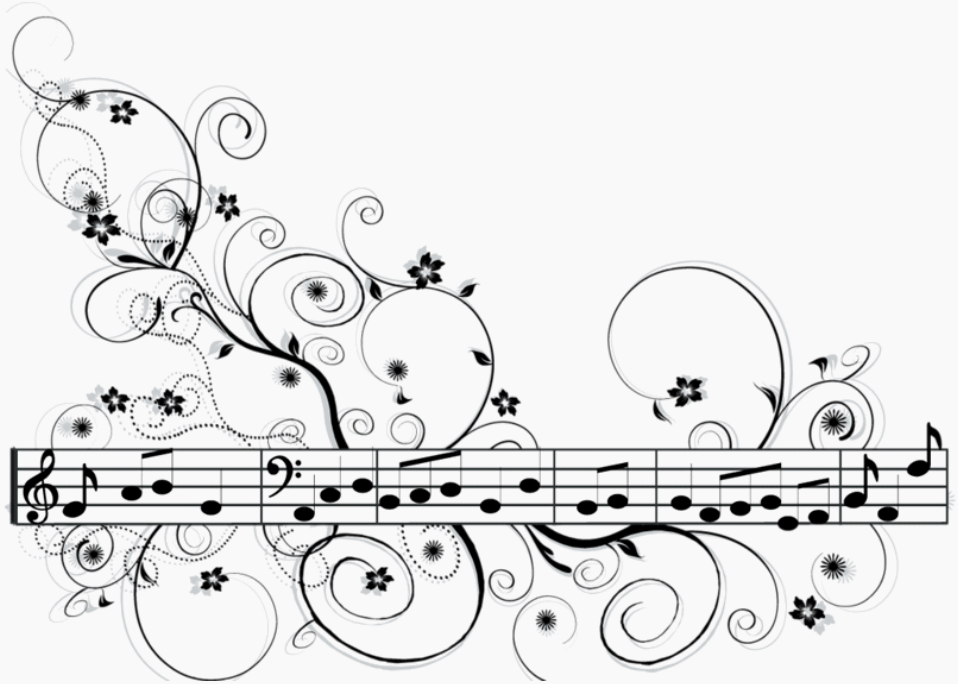
Opérateur	Rôle	Exemple	Résultat
=		`nombre = 8;`	8
+=	Addition	`nombre += 7;`	15
-=	Soustraction	`nombre -= 6;`	9
*=	Multiplication	`nombre *= 5;`	45
/=	Division	`nombre /= 4;`	11.25
%=	Reste (modulo)	`nombre %= 3;`	2.25

Incrémenter/décrémenter une variables

- Il est également possible d'augmenter ou de diminuer la valeur d'un nombre avec les opérateurs **++** et **--**
- Ce sont des **opérateurs d'incrémentations**, car il permettent d'incrémenter (augmenter de 1) la valeur d'une variable et inversement

```
let b = 0;          // b contient 0
b += 1;             // b contient 1
b++;                // b contient 2
console.log(b); // 2
```

Portée des variables



Portée des variables - Scope

- On appelle **portée** (**scope**) d'une variable la portion du code source dans laquelle cette variable est visible et donc utilisable.

- Les variables déclarées avec **let** et **const** ont une portée de type **bloc** : elles ne sont visibles qu'au sein du bloc de code dans lequel elles sont déclarées (ainsi que dans tous les sous-blocs éventuels).

- En JavaScript et dans de nombreux autres langages, un bloc de code est délimité par une paire d'accolades ouvrante et fermante

Portée des variables - Scope

```
let num1 = 0;
{
  num1 = 1; // OK : num1 est déclarée dans le bloc parent
  const num2 = 0;
}
console.log(num1); // 1
console.log(num2); // Erreur : num2 n'est pas visible ici !
```

La notion d’expression

La notion d’expression

- Une **expression** est un morceau de code qui produit une valeur.
- On crée une expression en combinant des variables, des valeurs et des opérateurs.
- Toute expression produit une valeur et correspond à un certain type.
- Le calcul de la valeur d'une expression s'appelle **l'évaluation**.
- Lors de l'évaluation d'une expression, les variables sont remplacées par leur valeur.

La notion d’expression

```
const c = 3; // 3 est une expression dont la valeur est 3
let d = c; // c est une expression dont la valeur est 3 (ici)
d = d + 1; // (d + 1) est une expression...
console.log(d); // 4
```

Expressions - priorités

- Une expression peut comporter des **parenthèses** qui modifient la priorité des opérations lors de l'évaluation.
- En l'absence de parenthèses, la priorité des opérateurs est la même qu'en mathématiques.

```
let e = 3 + 2 * 4; // e contient 11 (3 + 8)
e = (3 + 2) * 4;   // e contient 20 (5 * 4)
```

Expressions - templates

- Le langage JavaScript permet d'inclure des expressions dans une chaîne de caractères lorsque cette chaîne est délimitée par une paire **d'accents graves** seuls ou backticks (`...`).
- Une telle chaîne est appelée un **modèle de libellé** ou template literal.
- A l'intérieur, les expressions sont indiquées par la syntaxe **${expression}**.

- On utilise souvent cette possibilité pour créer des chaînes intégrant des valeurs de variables.

Expressions - conversion

```
const country = "Suisse";
console.log(`Je vis en ${country}`); // "Je vis en Suisse"

const x = 3;
const y = 7;
console.log(`${x} + ${y} = ${x + y}`); // "3 + 7 = 10"
```

Expressions - conversion

- L'évaluation d'une expression peut entraîner des conversions de type.
- Ces conversions sont dites **implicites** : elles sont faites automatiquement, sans intervention du programmeur.
- Par exemple, l'utilisation de l'opérateur **+** entre une valeur de type chaîne et une valeur de type nombre provoque la concaténation des deux valeurs dans un résultat de type chaîne.

```
const f = 100;
console.log("f contient " + f); // "f contient 100"
```

Expressions - conversion

- Le langage JavaScript est extrêmement tolérant au niveau des conversions de type.
- Cependant, il arrive qu'aucune conversion ne soit possible. En cas d'échec de la conversion d'un nombre, la valeur du résultat est **NaN** (Not a Number).

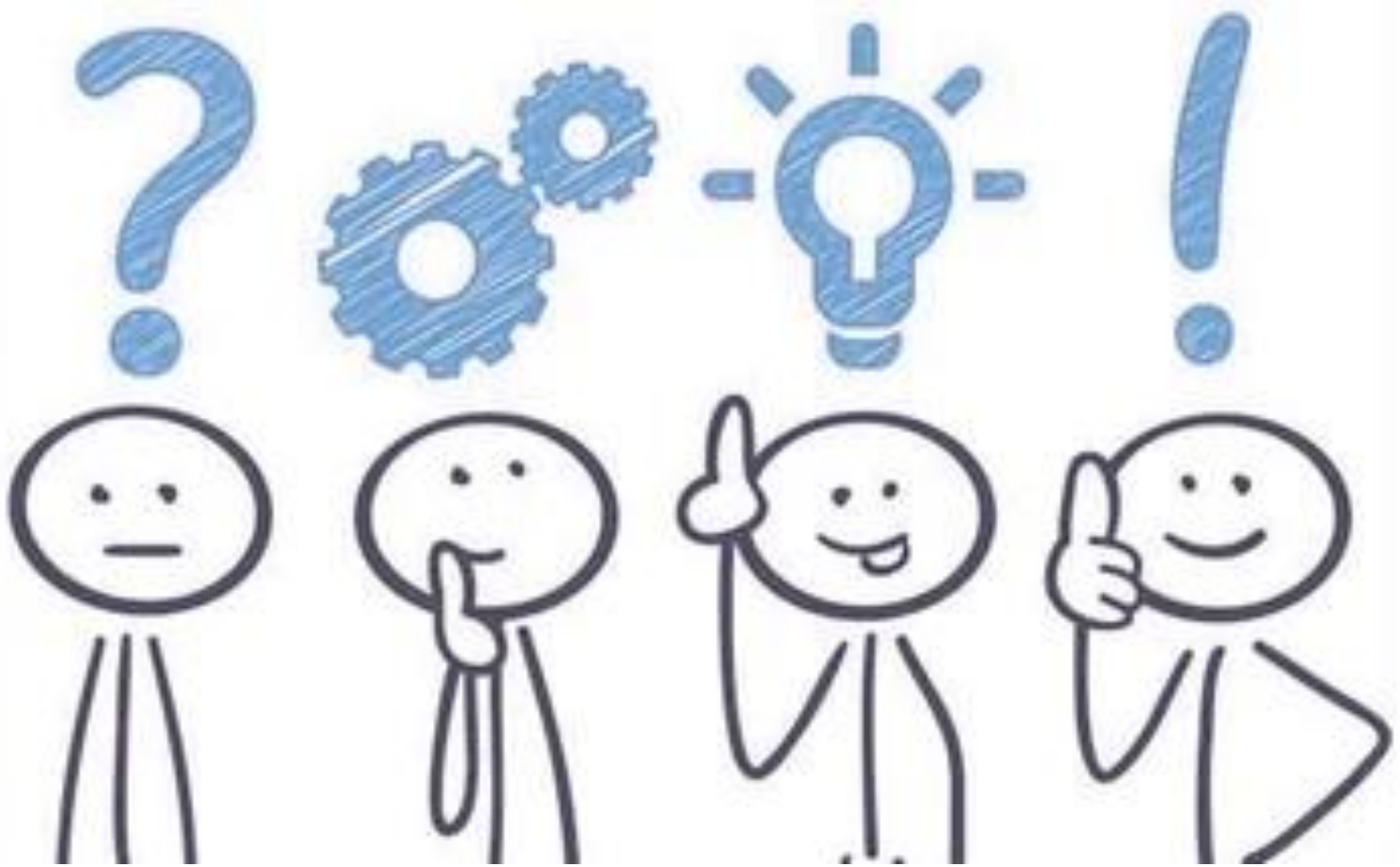
```
const g = "cinq" * 2;
console.log(g); // NaN
```

Expressions - conversion

- Il arrive parfois que l'on souhaite forcer la conversion d'une valeur dans un autre type.
- On parle alors de conversion **explicite**.
- Pour cela, JavaScript dispose des instructions **`Number()`** et **`String()`**

```
const h = "5";
console.log(h + 1); // Concaténation : affiche "51"

const i = Number("5");
console.log(i + 1); // Addition numérique : affiche 6
```



Exercices

- Exercice 2.3 : **Variables**
- Exercice 2.3 : **Questions de contrôle**